Linguaggi e Traduttori
a.a. 2005/2006
Tesina n° 1

Titolo: Aspect C
Descrizione

Negli ultimi anni ha visto una notevole crescita la programmazione orientata agli Aspect. Una delle caratteristiche fondamentali consiste nel ricercare nel codice delle funzioni per sostituirle con un'altra implementazione o per anteporre o posporre del codice, scelto dal programmatore.
In ingresso al traduttore sono forniti un file di direttive con una determinata sintassi e un file, o una lista di file C.
Il programma in primo luogo dovrà interpretate il file delle *direttive* che avrà la seguente sintassi:

fz <nome_fz> (substitute | ((in | out) (before | after))) (call by <nome_fz>)? {
(var{/ codice scelto dall'utente per variabili */})?*
/ codice scelto dall'utente */*
}

dove:
fz : è un identificativo che indica che verrà intercettata una funzione
<nome_fz> : rappresenta il nome della funzione da intercettare
substitute : dichiara che deve essere sostituito tutto il corpo della funzione
in before : Il codice viene inserito dentro la funzione prima di qualsiasi altro codice ma appena dopo la dichiarazione delle variabili
in after : Il codice viene inserito alla fine della funziona appena prima della keyword return se esiste
out before : Il codice viene inserito appena prima della chiamata alla funzione
out after : Il codice viene inserito appena dopo la chiamata alla funzione
call by <nome_fz> : Il codice viene inserito solo se la funzione è stata richiamata dalla funzione specificata in *<nome_fz>*. L'opzione *call by* può essere utilizzata solo con la direttiva *out* e non con *in*.
var : Se la direttiva var è presente, inserisce del codice appena dopo la dichiarazione delle variabili e comunque prima del codice inserito dall'aspect nel caso di direttiva *in before.*

Dopo aver letto gli aspect presenti nel file delle direttive e creato delle strutture in memoria per memorizzarli, dovrà essere lanciato un parser in grado di riconoscere un programma C e fare le sostituzioni necessarie dichiarate nel file delle direttive.

Consiglio: per la grammatica del C conviene cercare in rete quella originale, senza mettersi a scriverne una completa che potrebbe essere un lavoro lungo e complesso.

Gradirei, nel caso esista una funzione java in grado di riconoscere le espressioni regolari ?!?, la possibilità di specificare *<nome_fz>* come espressione regolare. Il programma non si complica, perché quando è analizzato il file delle direttive, anziché salvare il nome della funzione potete salvare direttamente l'espressione regolare. Quando invece parsificate il file C, anziché fare una comparazione tra il nome della funzione dichiarata nel file delle direttive e il nome della funzione reale, potete usare la funzione che vede se il nome della funzione è riconosciuta dall'espressione regolare o meno, in caso affermativo applicare le varie trasformazioni. In ogni caso non è obbligatorio, ma renderebbe il programma più utile per utilizzi reali.

Assieme al programma dovrà essere allegata una breve relazione contenente una veloce descrizione delle strutture dati utilizzate, di come il programma funziona e delle specifiche/limiti di funzionamento. Dovranno poi essere allegati dei **file di esempio** per verificare il corretto funzionamento del programma.

Esempio:

Dato il seguente file di direttive:

```
/* Primo aspect */
fz pippo out before call by pluto  {
        var { int a; }
        printf(a);
}

//Secondo aspect
fz pippo out after {
        var { int c; }
        printf("E' stata chiamata la funzione pippo");
}

//Terzo aspect
fz pippo in after {
        printf("Sono alla fine della funzione pippo");
}

fz asterix substitute {
        var { int c; }
        char b;
        printf("\n");
        pippo();
        printf("\n");
        printf("\n");
        pippo();
        printf("\n");
}
```

E il seguente file C:

```
pippo(){
        float a;
        char b;
        printf("\n");
        pippo();
        printf("\n");
        return;
}


pluto(){
        float a;
        char b;
```

```
        printf("\n");
        pippo();
        printf("\n");
}

asterix(){
        float a;
        char b;
        printf("\n");
        pippo();
        printf("\n");
}

obelix(){
        float a;
        char b;
        printf("\n");
        pippo();
        printf("\n");
}
```

Il traduttore dovrà generare il seguente file:

```
pippo(){
        float a;
        char b;
        printf("\n");
        pippo();
        printf("\n");
        printf("Sono alla fine della funzione pippo");
        return;
}


pluto(){
        float a;
        char b;
        int a;
        int c;
        printf("\n");
        printf(a);
        pippo();
        printf("E' stata chiamata la funzione pippo");
        printf("\n");
}

asterix(){
        int c;
        char b;
        printf("\n");
        pippo();
        printf("\n");
```

```
        printf("\n");
        pippo();
        printf("\n");
}

obelix(){
        float a;
        char b;
        int c;
        printf("\n");
        pippo();
        printf("E' stata chiamata la funzione pippo");
        printf("\n");
}
```